Proceedings of the 2013 International Conference on
Advanced Mechatronic Systems, Luoyang, China, September 25-27, 2013

A real-time walking robot control system based on Linux RTAI

Deng Liming, Zhao Xianchao
State Key Laboratory of Mechanical System & Vibration
Shanghai Jiao Tong University
Shanghai, China
dengdawn@sjtu.edu.cn, xczhao@sjtu.edu.cn

Qi Chenkun, Gao Feng
State Key Laboratory of Mechanical System & Vibration
Shanghai Jiao Tong University
Shanghai, China
chenkqi@sjtu.edu.cn, fengg@sjtu.edu.cn

***Abstract*— This study developed a real-time control system for a walking robot. The control system for the walking robot should have a real-time operating system, a small size, and extendable IO cards. The PC104 computer is selected due to its small size, reliability and availability of many IO cards. The Linux RTAI is selected because it is an open-source, efficient and hard real-time operating system. The Turbo PMAC PC104 card is used to control motor drivers because its small size, multi-axis synchronization and powerful control functions. Compared with CAN or EtherCAT bus control scheme, this system can easily support motor drivers from different companies. The software is reusable to different robots due to its independence on communication bus protocols and motor drivers. The motor control experiments are provided to show the satisfactory real-time control performance.**

Keywords—walking robot control; real-time control; PMAC; Linux RTAI; motion control

I. Introduction

Walking robot, with its flexibility, multi-DOF, better adaptive to unstructured environment, is able to access virtually 100% of the earth's land surface[1]. Although many research works in this field have been done, scientists are still interested, especially after the Bigdog[2] created by Boston Dynamics, Inc shows its high performance and great potential to the world. Motion control system is the fundamental element to the robot system if motors are chosen as actuators. Almost all the actions of the robot are finally attributed to the movement of the motors, which are implemented by the motion control system.

Motion control system can be mainly classified into two categories according to the communication method with motor amplifiers.

- Bus based communication control method. In this control system, algorithms are all integrated into the computer. And the programming need to call a lot of functions from the library provided by motor amplifiers, making the program dependent too much on motor amplifiers. If the motor amplifiers are changed, lots of work should do to change the huge code, which limits the application of this motion control system. The computer communicate with amplifiers via bus, such as CAN bus or EtherCAT bus. The bus communication can reduce the connective lines to make the system concise and clear for the connection. Figure 1 shows this structure.

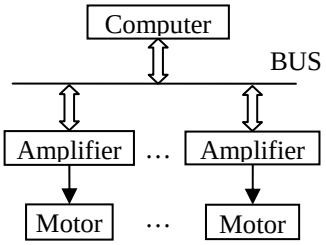

Fig. 1 The structure of bus based communication motion control method

- Embedded computer with motion control card. Amplifiers in this system are not directly connect the computer via bus, but connect with the motion control card. While the motion control card connect the computer through standard bus. The computer and controller combination can extend its high performance of calculation in computer and make full use of the controller's servo control ability. And this control system is friendly to many different kinds of amplifiers. Figure 2 shows the typical structure of this control system.

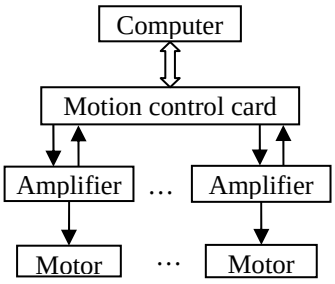

Fig. 2 Structure of the computer with motion control card method

Through the analysis above, both of the two method are feasible for the walking robot.

This work is partially supported by the National Natural Foundation of China (Grant No. 51175323), and National Program on Key Basic Research Project (Grant No. 2013CB035501).

978-0-9555293-9-9/13/$25.00

Here, we chose the PC and control card as the motion control system to avoid programming much in low level servo control and for its friendly support to different kind of amplifiers. Linux kernel patched with RTAI, as an open source real-time system, is selected to run in our CPU module. And the control card is Turbo PMAC PC/104 due to its small, compact and highly performance in servo control.

This paper proposes an implementation of the real-time motion control system for walking robot. Section I is the introduction of this paper, section II introduces the hardware about the system, section III gives a brief introduction about the software of this control system, section IV provides the experiment in the testing of its deterministic timing and also the motor move trajectory, and section V presents the conclusion.

II. Control system hardware

A. Architecture of the control hardware

This control system mainly composed with a motion controller Turbo PMAC, provided by Delta Tau, connected with a industrial PC, and also several amplifiers and motors. To make the system more complementary, many sensors are also added. The industrial PC responds to the high level planning and calculation, such as the trajectory generation, status monitoring, and processes sensor information. Turbo PMAC receives the data from the host PC and control motors by sending command to amplifiers. Figure 3 shows the general hardware architecture of the control system. Figure 4 shows the picture of the real CPU module and Turbo PMAC.

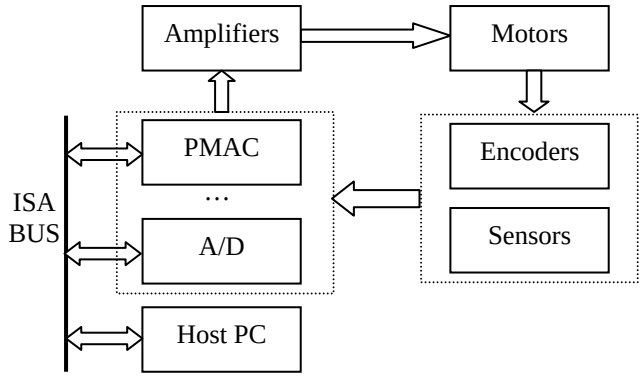

Fig. 3 The architecture of the control system

B. PMAC motion control system

B.1 Establish the PMAC system

As the Fig.3 shows, the host computer connect with the Turbo PMAC via ISA bus, other cards with this bus such as A/D card also can be extended in the system. Each Turbo PMAC can control four motors. With the help of ACC-1P (an expansion of the PMAC), the Turbo PMAC can control up to 12 motors which can cover the requirement of our walking robot.

Encoders installed on motors give the feedback both to Turbo PMAC and the amplifiers via shield twisted-pair cable, which can improve the anti-interference ability and guarantee the stable of the whole system. Amplifiers receive the analogy signal from Turbo PMAC and supply the power to motors via flat cable. Other sensors directly connect to the Turbo PMAC and also with the A/D card.

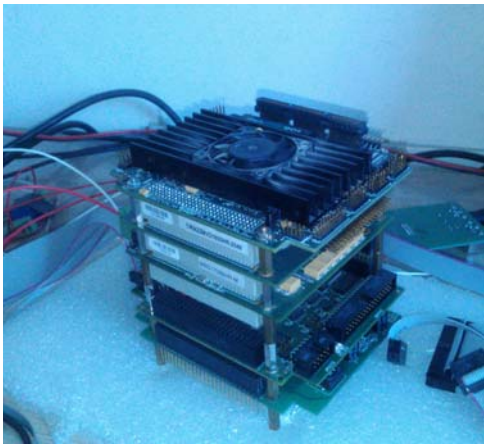

Fig. 4 The host PC and Turbo PMAC

After installed the system, test the system to ensure the connective correctly is necessary according to the Turbo PMAC[3] User Manual's instruction. Then a lot of stuff, such as frequency, move parameters and .etc, should configure first before make motors rotate well.

Another important thing is that tuning the PID parameters. Turbo PMAC provide a PID algorithm, which by far is the most commonly used servo filter[3]. Turbo PMAC also provides extended servo algorithm for more difficult dynamics and allows the users create their own servo algorithm for special occasions. Here, we use the PID servo algorithm, and tuning the parameters by using the PmacTuningPro2, a tool provided by Delta Tau Corporation.

B.2 Test the PMAC system

After finishing the connection and configuration of the system, it is nature to test its performance. Turbo PMAC provides powerful and flexible trajectory interpolation algorithm, such as LINEAR mode, SPLINE mode, also PVT mode and so on, which provide customers great autonomy to choose what their want according to their own situations.

Here, LINEAR mode and PVT mode are chose to test, because LINEAR mode is relative simple and PVT mode will be adopted in our system for its accuracy and excellent contouring capability. The test program is created in the Pewin32Pro2, an integrated development environment, under the instruction of the software manual[4] and user manual.

In the test program, the motor is required to move a cosine function, with 100 position interpolation, the time of each position interval is fixed. However, the LINEAR mode has acceleration and deceleration time at the first and the end point, while the PVT mode enwrap acceleration and deceleration time in the fixed time, thus the LINEAR mode takes a little

978-0-9555293-9-9/13/$25.00

more time than the PVT mode. Figure 5 and Figure 6 show the result of following required trajectory in two modes.

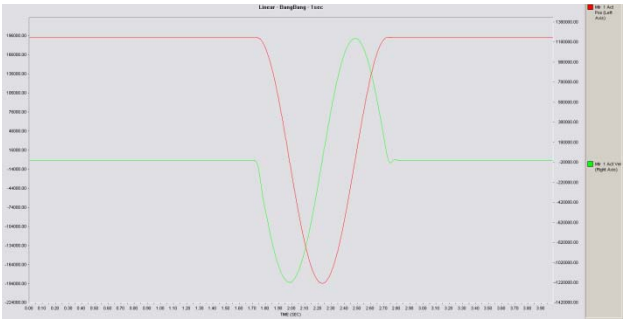

Fig. 5 Position and velocity profiles in LINEAR mode

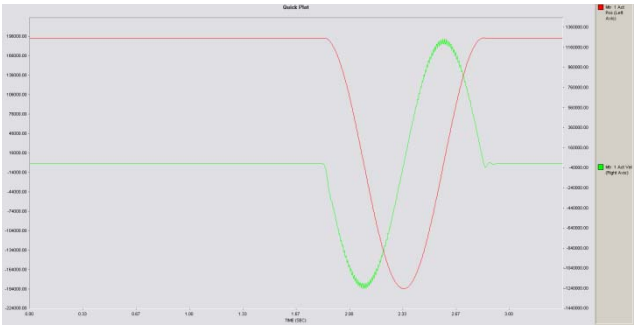

Fig. 6 Position and velocity profiles in PVT mode

In the upper figures, the red profile represent the position and the green profile represent the velocity during the motion. It shows that motors controlled by this system have good following capabilities and little vibration. At this point, the connection and the essential configuration for the PMAC have been successfully finished. Thus, we can turn to the next part to discuss the real-time operating system and the application program.

III. Real-time control software

A. Architecture of the control software

In the real-time system, the control software is mainly consist of four kinds threads, as shows in Figure 7. The main thread is response for initial and delete all the other threads. The signal gathering threads are trying to gather the signal from different sensors in a fixed frequency. Some of these kind threads send part of the data to remote GUI for monitoring, while others collecting the key information from the sensors and sending data to trajectory planning thread. The trajectory planning thread is the core of the control software. On one hand, it incorporate all the relative sensors' signals and process them as desired. On the other hand, it also can receive and response the commands from the remote GUI. The trajectory planning thread is aim to generate the desired positions or velocities for motors, and these data will be sent to the PMAC for control motors to targeted position by command sending thread. The arrow with straight line in Figure 7 means that the time for this process should be deterministic and guaranteed.

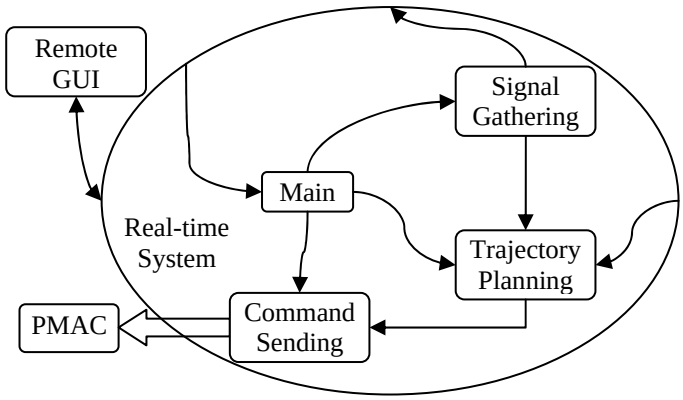

Fig.7 The architecture of the control software

B. Linux and RTAI

A real-time computer system can be defined as a system that performs its functions and responds to external, asynchronous events within a specified amount of time[5]. The real-time system can be divided into soft and hard real-time system. A Linux system can be regarded as a soft real-time system, while the hard real-time system should guarantee the deadlines critically. Many task such as rocket launching control should guarantee the time, much delay will cause a disaster.

There are many real-time operating systems available today, such as QNX, VxWorks, RTLinux, RTAI, Xenomai, Linux. QNX and VxWorks are commercial real-time system, RTLinux also has patent limitations. Xenomai and RTAI patched in Linux kernel are both free and open source project and can represent a validate alternative for the commercial real-time system. While the RTAI proved to be has slightly better performance than Xenomai[6]. Here, Real Time Application Interface (RTAI), as an open, supported, standard, efficient and free system with hard real-time capability, is chosen to control our robot.

RTAI is integrated into Linux through a text file containing a set of changes to its kernel source code, known as a patch, and a series of add on programs expanding Linux to hard real time by installing a generic Real Time Hardware Abstraction Layer (RTHAL)[7]. Figure 8 shows the simplified structure of the real-time system.

The RTHAL enwraps all the hardware interrupts and dispatches them to standard Linux or to real-time tasks depending on the requirements of the RTAI schedulers.

The RTAI distribution includes three different priority based, pre-emptive real time schedulers: the Uni-Processor (UP) scheduler; the Multi Uni-Processor (MUP) scheduler; and the Symmetric Multi-Processor (SMP) scheduler. It is generally not necessary for the user to manually install the proper scheduler because the installation process is usually able to determine the appropriate scheduler from the hardware configuration of the target machine. It then copies and links the appropriate scheduler so that it is called by the generic rtai_sched reference.

978-0-9555293-9-9/13/$25.00

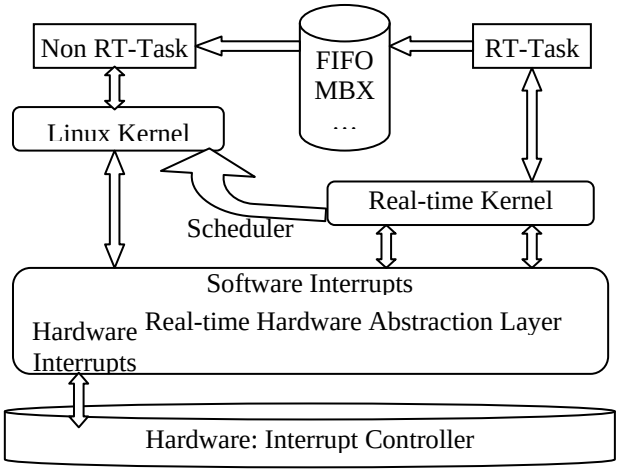

Fig. 8 The structure of the Linux RTAI

The scheduling policy for RTAI are EDF(Earliest Deadline First), FIFO(First In First Out), RR(Round Robin), RMS(Rate-Monotonic Scheduling). The FIFO is the default scheduling policy for real time task. While Linux scheduler defaults to SCHED_OTHER , which is a time-sharing policy most commonly used by Linux processes. SCHED_FIFO used to gain a greater level of control over the way in which the process is scheduled. Using this policy allows the process to preempt any other process scheduled using SCHED_OTHER and any process with a lower priority.

RTAI enable the hard real time task both in kernel space and user space. The two strategies offer both positives and negative aspects[8]. For the user mode is more simple and convenient to implement. RTAI's services are provided by several modules, which allow hard real-time, fully preemptive scheduling based on a fixed priority scheme. The some modules are[9]:

- rtai - the basic RTAI framework, plus interrupt dispatching and timer support.
- rtai_sched – the real-time, pre-emptive, priority-based scheduler, chosen according to the hardware configuration.
- rtai_fifos – FIFOs and semaphores.
- rtai_shm – shared memory.
- lxrt – LXRT for user space

With these modules, some related functions can be called to implement the hard real-time task.

After loading the modules, there is no much difference to program it with respect to general process, except calling the special function provided by RTAI to make it hard real-time. Be sure to start the timer once in a process. Then lock the memory, initial the task and make it to real-time task. The hard real-time task should not include a lot of task. The performance of hard real-time task can be guaranteed as few as possible. More hard real-time tasks may change them into soft real time. More detail instructions about how to program in Linux RTAI can be found in the DIAPM RTAI programming guide[9].

IV. Motion Control Experiments

In the control system, the rotary buffer provided by Turbo PMAC is used to receive the command lines generated from the host computer. And motors move as the rotary buffer instructed. If the motor finished to execute one command, the command line stored in the rotary buffer will be discarded leave new room for new command line. What we want to guarantee is that the command lines in rotary buffer should not too many, or the control object (here is robot) will have a long delay in interacting with the environment. And at least one command line stored in case to avoid motor stop without the command. These sending and consuming mechanism is shown in Figure 9.

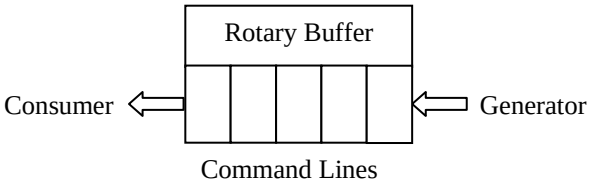

Fig. 9 Generating and consuming mechanism

Figure 10 shows the core program flow chart programmed in the upper computer. This thread should run in hard real-time to make it always have highest priority and not preempted by other non real-time task. Other threads such as display the motor status or print the progress of the real-time task are not showed in the chart.

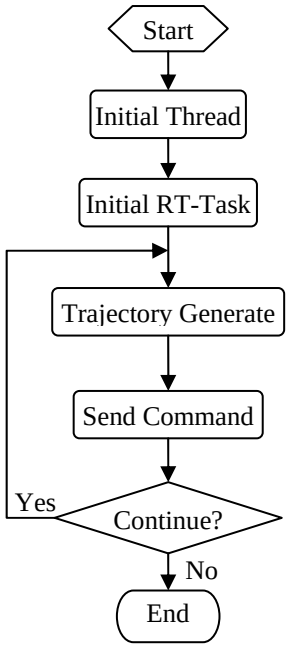

Fig. 10 The core program flow chart

In the motion control, PVT mode is chosen and set the time to 10 millisecond. It means the motor execute one command line will takes 10 millisecond. Thus, within this time, the host computer first should generate the desired

978-0-9555293-9-9/13/$25.00

position and velocity and then send it to the rotary buffer. The generation takes very few times about 50 micro second. However, when sending the command to rotary buffer, the time cannot be guaranteed.

TABLE I shows 10 longest times in sending the command to rotary buffer. The first column indicates in which loop it happens, and the second column shows the exacting time in sending the command, and the last column shows the whole loop takes to finish the task.

TABLE I. TEN LONGEST SENDING TIME IN NON-REALTIME DRIVER TEST

No.	Send time(us)	The Loop time(us)
14702	99194.982	99950.93
29934	92281.577	93071.001
29935	2533.993	4961.279
29936	5393.154	6323.824
45404	89387.038	90140.051
45406	2400.429	4816.463
47326	3851.072	19986.412
47426	3469.209	19978.858
62921	84512.758	87151.302
78209	100845.156	102220.363

In this test, it continues half an hour and about 90000 cycles in total, two command lines are send in each loop, and the hard real-time task is executed in periodic mode, set 20 milliseconds a cycle. It is obvious that the time cannot be guaranteed. In the 78209th loop, the sending time even exceed one hundred milliseconds, which is not allowed in the control system.

This happens because the driver of the PMAC is not a real-time driver. After a hard experience changing it to real-time driver, the deterministic time can be guaranteed. TABLE II also shows the ten longest times in sending the command to rotary buffer after change the driver to real-time type.

TABLE II. TEN LONGEST SENDING TIME IN REALTIME DRIVER TEST

No.	Send time(us)	The Loop time(us)
34473	324.045	19988.128
49467	323.973	19988.278
56342	328.384	19988.11
63245	324.345	19988.335
70134	328.062	19987.937
77123	324.278	19988.387
84240	328.162	19988.32
84664	324.255	19988.358
84833	328.654	19988.406
84909	324.483	19988.445

From TABLE II, we can see that the sending time is very small, and the whole loop time also can be guaranteed within 20 milliseconds.

A small problem should be paid attention is that the timer in the host computer is not exactly the same with the PMAC, thus before put the system into use, calibrate the timing is necessary. To be frank, using the interrupt would be a more elegant way to solve the problem fundamentally.

V. Conclusion

The aiming of the reported work is trying to implement a real-time motion control system for walking robot. The RTAI provide a good solution to the real-time operating system to guarantee the algorithm calculation time before deadline. With the extension of the real-time driver for Turbo PMAC, the communication between PMAC and host computer is also have the deterministic time. The PMAC has good performance to control the motors both in accuracy and fast response. Lots of tests have proved that the time from the algorithm calculation to motors execution can be guaranteed. And this control system can satisfy the requirement of the complex walking robot both with fast response and quick interaction with the environment.

Acknowledgment

During the time in implementing this real-time control system, many technique supports from Delta Tau help us configure the PMAC and also its programming, and Paolo Mantegazza help us modify the PMAC driver to real-time type, which make it possible to guarantee the time in the communication. And Giulio Cerrtui, also help us to create a lot of code.

References

[1]. Zico Kolter J, Ng A Y. The Stanford LittleDog: A learning and rapid replanning approach to quadruped locomotion[J]. The International Journal of Robotics Research, 2011, 30(2): 150-174.
[2]. Raibert M, Blankespoor K, Nelson G, etc. BigDog, the Rough-Terrain Quadruped Robot [J]. The International Federation of Automatic Control, 2008.
[3]. DELTA TAU Data Systems I. USER MANUAL for Turbo PMAC[M]. 2008.
[4]. DELTA TAU Data Systems I. SOFTWARE REFERENCE MANUAL for Turbo PMAC/PMAC2 [M]. 2010.
[5]. Furht B, Grostick D, Rabbat G, etc. Real-Time UNIX System: Design and Application Guide[M]. Kluwer Academic Publishers, 1990.
[6]. Barbalace A, Luchetta A, Manduchi G, etc. Performance comparison of VxWorks, Linux, RTAI and Xenomai in a hard real-time application[A]. In Real-Time Conference, 2007 15th IEEE-NPSS[C], 2007: 1-5.
[7]. Dozio L, Mantegazza P. Real time distributed control systems using RTAI[A]. In Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on[C], 2003: 11-18.
[8]. Racciu G, Mantegazza P. RTAI User Manual 3.4[M]. October, 2006.
[9]. Mantegazza P, Bianchi E, Angelo M, etc. DIAPM RTAI programming guide 1.0[M]. 2000.

978-0-9555293-9-9/13/$25.00